

Программирование Python

Лекция 5

Кафедра ИВТ и ПМ

2017



План

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

- Операции со строковым типом

- Форматирование строк

Типизация



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Прошлые темы

- ▶ Что такое модуль? Что такое пакет?
- ▶ Как подключить пакет?
- ▶ Что такое имя(идентификатор)?
- ▶ Что такое литерал?
- ▶ Что такое выражение?



Прошлые темы

Какую работу выполняют следующие функции?

- ▶ `abs()`
- ▶ `sleep()`
- ▶ `float()`
- ▶ `type()`
- ▶ `help()`



Прошлые темы

- ▶ Как работает тернарный условный оператор?



Прошлые темы

- ▶ Как работает тернарный условный оператор?
value1 **if** condition **else** value0
- ▶ Что такое генератор списка?
- ▶ Что такое кортеж?
- ▶ Что такое кортежное присваивание?
- ▶ Как обменять значения переменных используя кортежное присваивание?



Прошлые темы

Что будет в списке?

```
l = [ x/2 for x in [3, 14, 15, 9, 26, 5] if x <15 ]
```



Прошлые темы

Что будет в списке?

```
l = [ x/2 for x in [3, 14, 15, 9, 26, 5] if x <15 ]  
[1.5, 7.0, 4.5, 2.5]
```

```
l = [ x/2 if x< 15 else 42 for x in [3, 14, 15, 9, 26, 5]
```



Прошлые темы

Что будет в списке?

```
l = [ x/2 for x in [3, 14, 15, 9, 26, 5] if x <15 ]  
[1.5, 7.0, 4.5, 2.5]
```

```
l = [ x/2 if x< 15 else 42 for x in [3, 14, 15, 9, 26, 5]  
[1.5, 7.0, 42, 4.5, 42, 2.5]
```



Прошлые темы

```
for i,j in [(1,2), (3,4), (5,6)]:  
    print(i*j)
```



Прошлые темы

```
for i,j in [(1,2), (3,4), (5,6)]:  
    print(i*j)
```

2

12

30



Прошлые темы

Как оптимизировать следующий код?

```
sum = 0
for i in range(1, 10**10, 2):
    sum += i
```



Прошлые темы

Как оптимизировать следующий код?

```
sum = 0
for i in range(1, 10**10, 2):
    sum += i
```

Использовать формулу суммы арифметической последовательности

$$\text{sum} = (1 + 10^{10} - 1) / 2 * (10^{10} / 1)$$

Перед кодированием алгоритма, необходимо упростить этот алгоритм.



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Проверка истинности

- ▶ Любое число, не равное нулю, или не пустой объект интерпретируется как истина
- ▶ Числа, равные нулю, пустые объекты и специальный объект None интерпретируются как ложь.
- ▶ Операции сравнения и проверки на равенство применяются к структурам данных рекурсивно.
- ▶ Операции сравнения и проверки на равенство возвращают значение True или False (которые представляют собой версии чисел 1 и 0)
- ▶ Логические операторы and и or возвращают истинный или ложный объект операнд.



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Проверка истинности

Falsy значения - значения, которые могут быть преобразованы в False

- ▶ None
- ▶ 0
- ▶ 0.0
- ▶ 0j
- ▶ []
- ▶ {}
- ▶ ()
- ▶ ""

```
L = []
```

```
if L:
    print("Список НЕ пуст")
else:
    print("Список пуст")
```



Проверка истинности

Falsy значения - значения, которые могут быть преобразованы в False

```
L = []  
  
if L:  
    print("Список НЕ пуст")  
else:  
    print("Список пуст")
```



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Строковые литералы

Строка - тип данных, значениями которого является произвольная последовательность (строка) символов.

Как в Python задаются строковые значения?



Строковые литералы

Строка - тип данных, значениями которого является произвольная последовательность (строка) символов.

Как в Python задаются строковые значения?

- ▶ В одинарных кавычках

```
'some text'
```

- ▶ В двойных кавычках

```
"some text"
```

- ▶ В трёх парах кавычек

```
"""some text"""
```

В чём отличие третьего способа задания строкового литерала от первых двух?



Строковые литералы

Строка - тип данных, значениями которого является произвольная последовательность (строка) символов.

Как в Python задаются строковые значения?

- ▶ В одинарных кавычках

```
'some text'
```

- ▶ В двойных кавычках

```
"some text"
```

- ▶ В трёх парах кавычек

```
"""some text"""
```

В чём отличие третьего способа задания строкового литерала от первых двух? В исходном коде строковый литерал заданный

третьим способом может располагаться на нескольких строках. При выводе на экран знаки переноса строки сохраняются.



Символы и их коды

В Python нет отдельного *символьного типа*.

Вместо переменных символьного типа используются строки единичной длины.

В памяти символы представлены своими кодами, но при печати вместо числа выводится ответствующий коду в Unicode таблице символ.

- ▶ Получить код символа

```
ord('a') # 97  
ord('b') # 98  
ord('@')  
64
```

- ▶ Получить символ по коду

```
chr(97) # "a"  
chr(98) # 'b'  
chr(64) # '@'
```



Символы и их коды

Как получить символ следующий за x в Unicode таблице

Символы и их коды

Как получить символ следующий за x в Unicode таблице

```
chr( ord ('x') + 1 )
```



Представление целых чисел строками

Как получить символ следующий за x в Unicode таблице

Представление целых чисел строками

Как получить символ следующий за x в Unicode таблице

```
chr( ord ('x') + 1 )
```



Строковые переменные

В Python строковый тип называется **str**.

Записать строковое значение в переменную

через строковый литерал

```
S = "Hello"
```

```
S = "123"
```

```
S = "3.1415"
```

```
S = "" # пустая строка
```

конвертируя другой тип в строку

```
S = str( 123 )      # '123'
```

```
S = str( 3.1415 )   # '3.1415'
```

```
S = str( True )     # "True"
```

```
S = str( 1 + 2j )    # '(1+2j)'
```



Представление целых чисел строками

- ▶ Написание числа в шестнадцатеричной системе счисления
`hex(15)` # `"0xf"`
- ▶ Написание числа в восьмеричной системе счисления
`oct(15)` # `"0o17"`
- ▶ Написание числа в двоичной системе счисления
`bin(15)` # `"0b1111"`



Операции со строками

Операция со строками похожа на операции со списками или кортежами. Ведь это тоже упорядоченные наборы значений.

`help(str)`

- ▶ Конкатенация

```
S = "четыре" + "слова" + "без" + "пробелов"  
"четыресловабезпробелов"
```

- ▶ Повторение

```
S = "Ха" * 3      # "ХаХаХа"
```

Оба способа можно комбинировать

```
S = "." * 3 + " - это многоточие"  
  
'... - это многоточие'
```



Операции со строками

- ▶ Получение длины строки

```
S = "йцукен"  
len(S)      # 6
```

- ▶ Доступ по индексу (смещению)

```
S[0]        # "й"  
S[2]        # "у"  
S[-3]       # "к"
```

- ▶ Срезы

```
S[:2]       # "йц"  
S[2:4]      # "ук"  
S[::-1]     # "некуцй"  
S[:2]       # "йуе"
```



Операции со строками

- ▶ Проверка вхождения подстроки в строку

```
'x' in "there is no X"      # False  
'x' in "there is no x"     # True  
'there' in "there is no x" # True
```



Операции со строками

- Подсчёт числа вхождений

S.count(sub[, start[, end]]) -> int

Возвращает число вхождений подстроки.

sub - подстрока

start - позиция начала поиска

end - позиция окончания поиска

start и end имеют такой же смысл, что и в срезе, т.о. поиск проходит ДО позиции end

```
S = "qwerty"
S.count("rty")           # 1
"0000".count("000")     # 1
"0000".count("00")      # 2
"0000".count("aaa")     # 0
```



Операции со строками

- Поиск подстроки

S.find(sub[, start[, end]]) -> int

Возвращает позицию подстроки в данной строке.

-1 - если подстрока не найдена.

sub - подстрока

start - позиция начала поиска

end - позиция окончания поиска

start и end имеют такой же смысл, что и в срезе, т.о. поиск проходит ДО позиции end

```
S = "qwerty"  
S.find("we")    # 1  
S.find("000")   # -1  
S.find("rty")   # 3
```



Операции со строками

► Проверка всех значений строки

В строке записаны только цифры?

```
"1231".isdigit()    # True
```

```
"12.31".isdigit()   # False
```

```
"12 31".isdigit()   # False
```

```
"-31".isdigit()     # False
```

В строке только буквы?

```
"qWerR".isalpha()   # True
```

```
"numb3rs".isalpha() # False
```



Операции со строками

- ▶ Проверка всех значений строки

Строка в нижнем регистре?

```
"qwекен".islower() # True
```

```
"123абв".islower() # False
```



Операции со строками

- Замена подстроки

S.replace(old, new[, count]) -> str

Заменяет подстроку *old* на *new* и возвращает новую строку.

count - число замен. Если не задано, то заменяется все подстроки.

Длины новой и старой подстрок могут не совпадать.

```
S = "qwerty"
```

```
S2 = S.replace("we", "*1*")
```

```
'q*1*rty'
```



Операции со строками

- ▶ `S.lower()` -> str
Возвращает новую строку, где все символы в нижнем регистре
- ▶ `S.upper()` -> str
Возвращает новую строку, где все символы в верхнем регистре



Операции со строками

- ▶ Разбить строку на несколько частей
S.split(sep=None, maxsplit=-1) -> list of strings
Возвращает список из строк.
sep - подстрока-разделитель
maxsplit - максимальное число элементов

```
s = "one two three"  
s.split()  
['one', 'two', 'three']
```

```
s.split()  
['onetwothree']
```

```
s = "oneABCtwoABCthree"  
s.split('ABC')  
['one', 'two', 'three']
```



Операции со строками

- ▶ "Склеивание" списка из строк в одну строку

S.join(iterable) -> str

iterable - набор строк, например список S - строка, которая буде вставлена в промежутки.

```
"; ".join( ['one', 'two', 'three'] )  
'one; two; three'
```

```
"".join( ['one', 'two', 'three'] )  
'onetwothree'
```



Операции со строками

- ▶ другие операции
см. `help(str)`



Операции со строками

В Python строки как и кортежи - неизменяемый тип данных. Это значит, что при изменении любой строки - создаётся новая строка.

Об этом стоит помнить при работе с длинными строками.

Как можно оптимизировать этот код?

```
S = ""  
for i in range(1000000):  
    S += str(i)
```



Операции со строками

В предыдущем случае, при каждой итерации цикла будет создаваться новая строка. Причём длины строк будут постоянно увеличиваться.

Лучше всего создать список из добавляемых элементов, а потом склеить их в одну строку.

```
L = []  
  
for i in range(1000000):  
    L += str(i)  
  
S = "".join(L)
```



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Форматирование строк

Для того чтобы подставить в строку числовые или другие значения может применяться метод **format**.

Этот метод работает для строки, которая задаёт *формат* для помещаемых в неё значений.

Позиция каждого значения в строке задаётся порядковым номером этого значения в параметрах format. Нумерация начинается с нуля.

```
"Скорость {0} равна {1} м/с".format( "самолёта", 250)
```

Вместо *0* будет подставлено *самолёта*

Вместо *1* будет подставлено *250*

В результате работы метода создаётся *новая строка*.



Форматирование строк

Число позиций в строке формата должно совпадать с числом параметров метода `format`

```
""""{0}° {1}' {2}" с.ш., {3}° {4}' {5}" в.д."""\n    .format(52, 2, 05.0, 113, 31, 44.9)
```

Результат:

52° 2' 5.0"с.ш., 113° 31' 44.9"в.д.

Символ обратной косой черты позволяет записать длинную операцию в две строки.



Форматирование строк

Представление целых в строках

```
"{0:5}".format(123)  # ' 123'
```

5 - число позиций для вывода числа.

После числа позиций можно указать вид числа d - десятичная форма h - шестнадцатеричная форма o - восьмеричная форма b - двоичная форма

```
"{0:5d}".format(15)  # '   15'
```

```
"{0:5x}".format(15)  # '    f'
```

```
"{0:5o}".format(15)  # '   17'
```

```
"{0:5b}".format(15)  # ' 1111'
```



Форматирование строк

Представление вещественных чисел в строках

```
"{0:7.3f}".format(3.14159) # ' 3.142'
```

```
"{0:7.3e}".format(3.14159 / 100) # '3.142e-02'
```

7 - число позиций для вывода числа.

3 - число знаков после запятой

f - вещественное число в десятичной записи e - вещественное число в экспоненциальной записи



Форматирование строк

Аналогично вещественным или целым числам можно управлять выводом подстрок

```
"{0:s}".format('abc')  # 'abc'
```

```
"{0:5s}".format('abc')  # '   abc'
```



Форматирование строк

Выравнивание > - по правому краю

< - по левому краю

—

```
"{0:5s}".format('abc')    # '   abc'
```

```
"{0:<5s}".format('abc')   # 'abc   '
```

```
"{0:5d}".format(12)       # '   12'
```

```
"{0:^5d}".format(12)      # '  12  '
```



Outline

Прошлые темы

Преобразование выражений к логическим выражениям

Преобразование выражений к логическим выражениям

Строковый тип

Операции со строковым типом

Форматирование строк

Типизация



Динамическая типизация

Что такое **динамическая типизация**?
Что такое **статическая типизация**?
Какая типизация используется в Python?



Динамическая типизация

Переменные – это записи в системной таблице, хранящей ссылки на объекты.

Переменная не содержит информации о типе.

Объекты – это области памяти с объемом, достаточным для хранения значений.

Именно объект содержит информацию о типе.

Ссылки – это указатели на объекты, которые работают как объекты (автоматически разыменовываются).

Когда переменная участвует в выражении, ее имя замещается объектом, на который она в настоящий момент ссылается.
В этом смысле про объекты можно думать как про значения.



Динамическая типизация

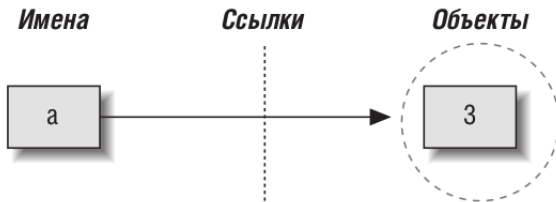
Что происходит при выполнении следующей операции?

`a = 3`

1. Создается объект, представляющий число 3.
2. Создается переменная `a`, если она еще отсутствует.
3. В переменную `a` записывается ссылка на вновь созданный объект, представляющий число 3.

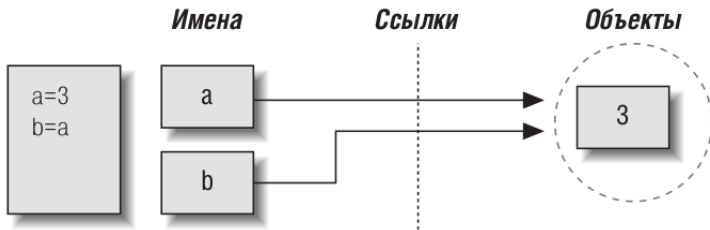


Динамическая типизация



Динамическая типизация

Две разных переменных могут ссылаться на один и тот же объект.



Такие переменные называются **разделяемыми ссылками**.



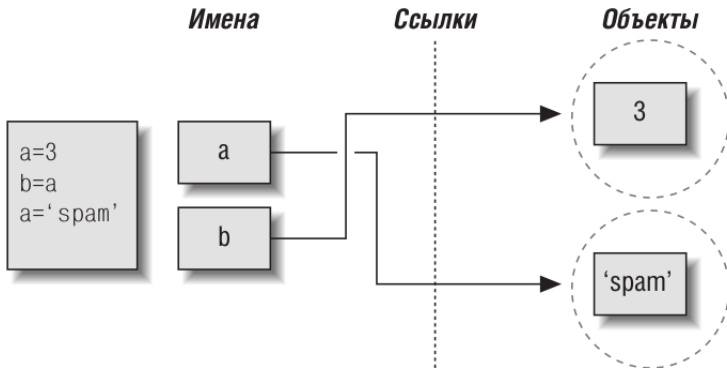
Динамическая типизация

Что будет если задать переменной **a** новое значение?



Динамическая типизация

Что будет если задать переменной **a** новое значение?



Динамическая типизация

Сколько переменных ссылается на данный объект (значение)?

getrefcount (значение) - возвращает число ссылок на данное значение.

```
from sys import getrefcount
```

```
>>> getrefcount(0)  
4993
```

```
>>> getrefcount(100)  
58
```

```
>>> getrefcount("a")  
338
```



Динамическая типизация

Python оптимизирует расход памяти для некоторых простых типов и не создаёт новое значение если оно уже есть.

Такая оптимизация называется **кэшированием**.

Например, создание трёх переменных с одинаковым целым значением не приведёт к увеличению расхода памяти в три раза:

```
a = 123456789012
b = a
c = b
getrefcount(a) # 4
```

"Лишняя" копия значения создаётся при передаче его в функцию `getrefcount`



Сборка мусора

В интерпретаторе Python используется **сборка мусора (garbage collection)** - освобождение памяти занимаемой неиспользуемыми значениями.

Этот механизм основан на подсчёте числа ссылок на объект. Причём если объект ссылается сам на себя, то это определит.



Изменяемые и неизменяемые типы

Все типы данных в Python можно разделить на **изменяемые** и **неизменяемые**.

При записи нового значения неизменяемого типа в переменную - старое значение не изменяется, а создаётся новое.

О старом значении позаботится сборщик мусора.

- ▶ Неизменяемые типы (immutable):
bool, int, float, complex, str, tuple, frozenset
- ▶ Изменяемые типы (mutable):
list, dict, set



Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming

